Fachhochschule Köln
Cologne University of Applied Sciences

Ereignisgesteuerte Systeme im Web

Seminararbeit

ausgearbeitet von

Benjamin Krumnow

betreut von

Prof. Dr. Kristian Fischer

vorgelegt an der

Fachhochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Medieninformatik

Gummersbach, im Juli 2012

Abstract

Informationen beim Eintreten eines Ereignis zu übermitteln, ist die Aufgabe ereignisorienterter Systeme. Für die Entwicklung solcher Systeme, gibt es eine Reihe von Technologien und Standards, welche je nach Plattform oder Anwendungskontext verschiedene Vorteile besitzen. Innerhalb dieser Seminararbeit werden die webbarsierten Systeme betrachtet, in dem auf die Konzepte von Light und Fat Ping, Publish-Subscribe sowie zentrale und dezentrale Architekturen eingegangen wird. Des Weiteren werden die Technologien XMPP, Googles C2DM, Amazon SNS und SQS, Twitter und Apples Push Notification beschrieben. Abschließend werden aus diesen Betrachtungen Merkmalsdimensionen abgleitet, die dabei helfen sollen, geeignete Lösungen in einem Anwendungskontext auszuwählen. Zusätzlich können diese bei der Konzeption von Systemen als Hilfestellung herangezogen werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Abkürzungsverzeichnis

Amazon SNS	Amazon Simple Notification Service
Amazon SQS	Amazon Simple Queue Service
APNs	Apple Push Notification Service
AWS	Amazon Web Service
C2DM	Cloud 2 Device Messaging
DNS	Domain Name System
FQDN	Fully Qualified Domain Name
HTTP	Hyper Text Transfer Protokoll
IAM	AWS Identity and Access Management (IAM)
IEFT	Internet Engineering Task Force
IP	Internet Protocol
JID	Jabber ID
JMS	Java Messaging Service
JSON	JavaScript Object Notation
RFC	Request for Comments
RSS	Really Simple Syndication
RSV record	Service Resource Record
SASL	Simple Authentication and Security Layer
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TLS	Transport Layer Security
XEP	XMPP Extension Protocol
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

1 Einleitung

In diesem Kapitel wird die Motivation und Zielsetzung für diese Seminararbeit dargelegt. Anschließend erfolgt eine Beschreibung des Vorgehens mit der das Ziel erreicht werden soll. Der letze Abschnitt umschreibt die einzelnen Kapitel.

1.1 Motivation

Architekturen im Web haben sich aufgrund steigender und wechselnder Anforderungen, welche u.a. durch neu entstandene Anwendungsfälle, sowie Anwendungskontexte oder stark anwachsende Nutzerzahlen verursacht werden, stetig verändert. Eine dieser Anforderungen ist es Daten bei verteilten Anwendungen aktuell zu halten und zwar so, dass die zeitliche Verzögerung möglichst minimal ist. Dies ist insbesondere bei zeitkritischen Anwendungen, die Daten in Echtzeit benötigen, erforderlich. Erreicht werden kann dies u.a. durch stetiges Abrufen der Daten, was unter dem Pull-Paradigma bekannt ist. Hierbei ist die Aktualität der Daten abhängig von der Abfragefrequenz, welche wenn sie erhöht wird, eine höhere Aktualität bietet, aber auch den Verbrauch von Strom, Datentransfer und Rechenlast steigert. Aus diesem Grund bietet sich die Verwendung des Push-Paradigmas an, da hier die Daten zugestellt werden, ohne diese stetig abfragen zu müssen. Dies impliziert, dass Daten in ereignisorientierten Systemen aperiodisch zugestellt werden, d.h dass Anwendungen benachrichtig werden, wenn neue relevante Daten existieren. In diesem Sinne wird diese Art der Kommunikation in verteilten Systemen Echtzeitkommunikation genannt.

In den letzen Jahren sind eine Reihe an Technologien entstanden, die diese Kommunikationsform für Anwendungen ermöglichen. Dies ist u.a. im Bereich der mobilen Plattformen wie Android, iOS, Windows Phone 7[1] und RIM OS[2], welche jeweils ihren eigenen Benachrichtigungsdienst entwickelt haben, zu beobachten. Aber auch plattformübergreifende Dienste und Technologie, wie Amazon SNS, Twitter, PubSubHubBub, XMPP und weitere sind mittlerweile verfügbar. Was diese Dienste bzw. Technologien unterscheidet und wie sie funktionieren, ist Gegenstand dieser Seminararbeit sein.

1.2 Zielsetzung

Die Zielsetzung dieser Arbeit ist es einen Überblick von verschiedenen aktuellen Technologien zu ermöglichen, die eine asynchrone Anwendung-zu-Anwendung Kommunikation in Echtzeit anbieten. Dabei sollen die Unterschiede der Technologien herausgestellt werden, so dass dessen Charakteristika ersichtlich werden.

[1] http://tinyurl.com/cgoncax
[2] https://developer.blackberry.com/services/push/

1.3 Vorgehen

Zur Erreichung der hier angestrebten Zielsetzung werden zuerst Begriffe und Techniken erläutert, die für Echtzeitkommunikationsverfahren relevant sind. Danach werden Kommunikationsstrukturen einiger dieser Verfahren beschrieben, wobei diese nach ihrer aktuellen Relevanz ausgewählt wurden. Der Vergleich dieser Verfahren und dessen Differenzierung soll erreicht werden, in dem Dimensionen abgeleitet werden, die zur Unterscheidung der Technologien herangezogen werden können

1.4 Übersicht der Kapitel

In diesem Kapitel wurde mittels der Motivation und Zielsetzung der grundlegende Gedanken für diese Seminararbeit aufgegriffen. In Kapitel 2 werden die Grundlagen behandelt, die essentiell für das Verständnis der weiteren Kapitel sind. In dem darauf folgenden Kapitel 3 werden einige ausgewählte Verfahren erläutert und mit den daraus gewonnen Erkenntnissen in Kapitel 4 Dimensionen identifiziert.

2 Grundlagen

In diesem Kapitel werden grundlegende Begriffe und Verfahren erläutert. Da ein wesentlicher Anteil, der in dieser Seminararbeit behandelten Technologien, nach dem Publish-Subscribe-Paradigma arbeitet, soll dieses hier beschrieben werden. Des Weiteren wird auf die Unterschiede von Fat und Light Ping eingegangen, da diese für das weitere Verständnis benötigt werden. Abschließend wird dargelegt, was unter eine zentralen bzw. dezentralen Architektur im Kontext dieser Arbeit zu verstehen ist.

2.1 Publish-Subscribe

Das Publish-Subscribe-Paradigma ermöglicht einen Nachrichtenaustausch, bei dem Sender und Empfänger entkoppelt werden und sich somit von dem klassischen Point-to-Point-Paradigma unterscheidet. Die Kommunikationsteilnehmer (bzw. Sender und Empfänger) bestehen aus Publishern (Anbietern), welche Events (bzw. Nachrichten) für andere Teilnehmer veröffentlich und Subscribern (Abonnenten), welche die für sie interessante Nachrichten oder Events abonnieren (siehe Abbildung 1). Die Subscriber werden benachrichtig, sobald ein Event von den Publishern veröffentlicht wird. Dabei muss der Publisher nicht die Subscriber kennen und ebenso wenig die Subscriber die Publisher, was im Gegesatz zu Point-to-Point steht, da hier der Sender den Empfänger adressiert. Um diese Entkopplung zu realisieren, ist ein Service erforderlich, der zwischen den Publishern und Subscribern vermittelt. Diese werden u.a. Message Broker oder Event Dispatcher genannt. Das Veröffentlichen von Events bzw. Nachrichten ist dabei asynchron, da nicht sicher ist, ob die Subscriber zu diesem Zeitpunkt erreichbar sind. Das heißt der Publisher sendet die Nachricht an den Service, ohne Gewissheit wann ein Subscribe die Nachricht erhält.

Es existieren unterschiedliche Ansätze, wie ein Subscriber sein Interesse für Events bzw. Nachrichten ausdrücken kann, wobei Eugster et al. [3] zwischen Topic-, Content- und Type-basierenden Ansätzen unterscheiden. Topics sind dabei Themen zu denen Events nach ihrem Inhalt gesendet werden. Das heißt Subscriber abonnieren die Topics zu denen sie Benachrichtigungen erhalten wollen. Content-basierende Verfahren selektieren Events nach ihren Eigenschaften. Die Eigenschaften können u.a. über Attribute, die ein Event besitzen kann, abgebildet werden. Die Subscriber können dann anhand von Filtern, die sie definieren, die Events abonnieren. Bei Type-basierenden Ansätzen abonnieren die Subscriber einen oder mehrere Typen von Objekten und erhalten dann alle Events dieses Typs.

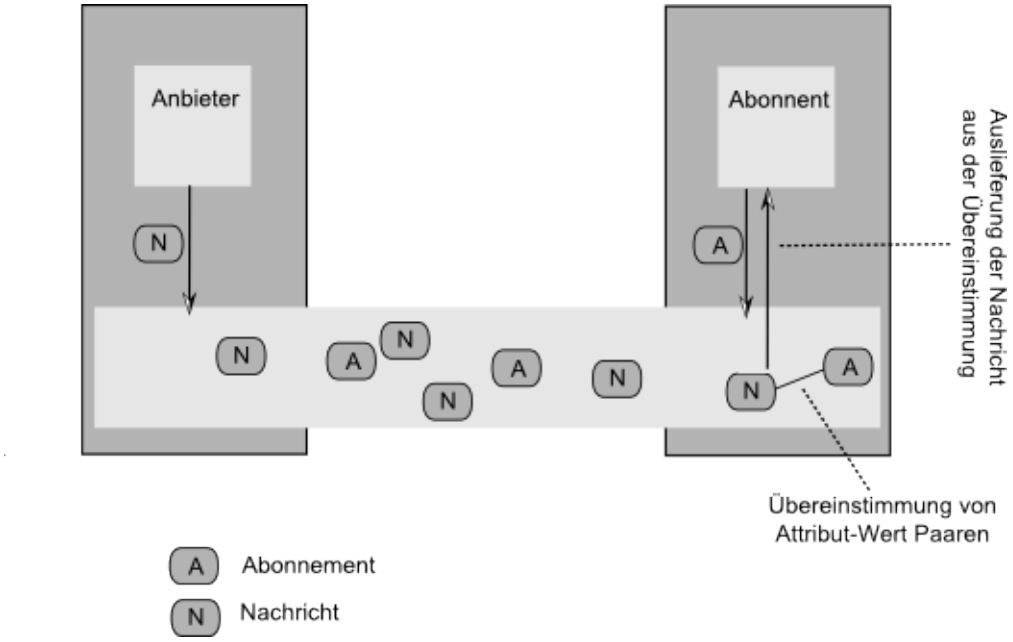

Abbildung 1: Schematische Darstellung des Publish-Subscribe Paradigmas nach [21]

2.2 Fat Ping und Light Ping

Die Benachrichtigungen die bei auf Event basierenden Verfahren übertragen werden, können nicht nur darüber Aufschluss geben, dass neue Daten zur Verfügung stehen, sondern den Nutzdatenanteil auch direkt beinhalten. Demnach werden Benachrichtigungen, die einen Nutzdatenanteil beinhalten als Fat Ping und ohne diesen Anteil als Light Ping bezeichnet. Bei der zweiten Variante liegt es im Ermessen des Empfängers, ob die Nutzdaten zusätzlich abgerufen werden (Beispielsweise über einen in der Benachrichtigung enthaltene URI), da dies in einigen Anwendungsfällen nicht erforderlich ist. Die Verwendung von Fat oder Light Pings ist von der Technologie abhängig, wobei einige Technologien beide Varianten unterstützen (siehe XMPP in Abschnitt 3.1). Zu beachten dabei ist, dass sich nach [10] die Wahl des Verfahrens auf die Bandbreite, Latenz und CPU-Auslastung auswirken kann.

2.3 Zentral und Dezentral

Webbasierte Services, wie Twitter oder Cloud 2 Device Messaging (C2DM) von Google, erscheinen jeweils aus einer externen Sichtweise wie ein zusammenhängendes System. Allerdings ist anzunehmen, dass zur Bewältigung verschiedener Aufgaben, wie etwa das Ver-

senden von Nachrichten, Authentifitzierung etc. unterschiedliche Maschinen bzw. Systeme zum Einsatz kommen. Ein weiterer Grund zur Verteilung von Aufgaben auf mehrere Instanzen kann eine angemessene Lastenverteilung sein. Bei einer internen Betrachtung dieser Dienste wird also eine dezentrale Architektur ersichtlich. Verschiebt man die Perspektive nach außen, wobei Anwendungen einen dieser Dienste nutzen, ergibt sich ein anderes Architekturbild, welches auch zentral sein kann, da beispielsweise alle externen Anwendungen über einen Dienst Daten austauschen.

Die Unterscheidung zwischen einer dezentralen und zentralen Architektur soll im weiteren Verlauf dieser Seminararbeit aus einer externen Perspektiven erfolgen. Daraus ergibt sich, dass Dienste wie Googles C2DM und Twitter als zentral angesehen werden und XMPP als dezentral, da hier keine zentrale Instanz (Beispielsweise in Form einer Organisation oder Unternehmen) erforderlich ist.

3 Beschreibung der Verfahren

In diesem Kapitel werden einige Technologien, die eine Echtzeitkommunikation ermöglichen, besprochen. Dabei liegen im Fokus der Betrachtung die Kommunikation und die Eigenschaften der Nachrichtenübertragung. Ausgewählt wurden die Technologien u.a. nach ihrer praktischen Relevanz und Bekanntheit.

3.1 XMPP

Das Extensible Messaging and Presence Protocol (XMPP) ist eine offene Technologie, die Echtzeitkommunikation ermöglicht und XML als Übertragungsformat verwendet [17, S. 3]. Dabei ist die Technologie XMPP eng mit der Bezeichnung Jabber verknüpft, da die Basis der Syntax und Semantik von Jamie Miller und der Jabber Open Source Community entwickelt wurde [16, Abschnitt 1.2]. Als die Kernprotokolle der Jabber Community bei der Internet Engineering Task Force (IEFT) vorgelegt wurden, entschied man sich dazu den Namen XMPP für das Protokoll zu verwenden, um dieses von der Community trennen zu können [17, S. XIII].

XMPP umfasst eine Reihe von Diensten mit denen eine Vielzahl an Anwendungsszenarien möglich sind, wie u.a. Instant Messaging, Gruppenchats, Voicechats, Videokonferenzen, Benarichtigungssysteme, leichtgewichtige Middleware, um nur einige zu nennen[3]. Diese Dienste sind durch Spezifikationen festgelegt, welche zum einen aus den Kernprotokollen und zum anderen aus Extensions bestehen. Die Kernprotokolle sind in den Request for Comments (RFC) spezifiziert und werden von der IEFT publiziert, während die Extensions in den XMPP Extension Protocols (XEP) bestimmt werden[17, S. XIII].

Diese Dienste bestehen u.a. aus übergreifende Kommunikationsfunktionen wie die Verschlüsselung der Übertragung und Authentifizierung, welche beide im RFC6120 angeben sind. Für die Verschlüsselung kann TLS und für die Authentifizierung das Simple Authentication and Security Layer (SASL)[4] Protokoll verwendet werden. Zur Umsetzung einiger Anwendungsszenarien, insbesondere Instant Messanger, stehen Dienste wie Präsenz, Kontaktlisten (beide RFC6121), Eins-zu-Eins-Kommunikation (RFC6120), Multiuser-Chat (XEP-0045) und Peer-to-Peer Media Sessions (XEP-0166), zur Verfügung. Die Präsenz zeigt den Status von Entitäten an, beispielsweise ob ein Nutzer Online oder Offline ist. Das Entdecken von angebotenen Diensten einer XMPP-Entität wird über das XEP-0030 spezifiziert. Ein weitere Dienst ist der Notification Service (XEP-0060), der eine Echtzeit-Kommunikation nach dem Publish-Subscribe-Paradigma, weshalb dieser später detaillierter

[3] Siehe hierzu http://xmpp.org/about-xmpp/
[4] siehe hierzu http://tools.ietf.org/html/rfc4422

betrachtet wird. Es stehen noch eine Reihe weiterer Dienste zur Verfügung, die über die hier beschriebenen Dienste hinaus gehen und der Literatur [17] entnommen werden können.

3.1.1 Aufbau und Grundlagen

XMPP benutzt eine dezentrale Client-Server-Architektur bei der jeder einen XMPP-Server betreiben kann. Die Clients[5], welche auf Anwendungsebene ebenfalls XMPP verwenden, kommunizieren dabei über XMPP-Server und in der Regel nicht direkt miteinander (siehe Grafik 2). Allerdings sind auch Peer-to-Peer Verbindung möglich, wie in [15] beschrieben wird. Sendet ein Client eine Nachricht an einen oder mehrere Empfänger, die nicht den gleichen XMPP-Server verwenden, so wird der XMPP-Server des Empfängers von dem XMPP-Server des Senders direkt angesprochen, ohne dass eine weitere Zwischenstation, wie beispielsweise bei E-Mail, verwendet wird. Die Verbindung zwischen mehreren XMPP-Servern ist hierbei optional, d.h. es ist auch möglich ein geschlossenes System zu betreiben. Für die Übertragung können verschiedene Protokolle verwendet werden[6], allerdings wird innerhalb dieser Seminararbeit nur die Übertragung mit TCP behandelt. Die Kommunikation ist dabei asynchron, welche über zwei persistenter XML-Streams mittels TCP erfolgt. Ein Kanal dient dabei zum Senden von Daten vom Client zum Server und ein Kanal für die entgegengesetzte Richtung. Gleiches gilt für die Server-zu-Server Verbindung.

Damit die Entitäten (z.B. Clientanwendungen, Bots und Server) im Netz kommunizieren können, benötigen sie Adressen, welche bei XMPP auf dem Domain Name System basieren und JabberID (JID) genannt werden. Serverseitig hat eine JID die Form einer Webadresse (z.B. example.org), während bei Entitäten, die über die Server kommunizieren, Accounts verwendet werden, die die Form einer E-Mailadresse (z.B. user@example.org[7]) besitzen. Die Adressen für diese Entitäten müssen global oder für das geschlossene Netz eindeutig sein, damit eine fehlerfreie Adressierung erfolgen kann. Dennoch können mittels eines Accounts mehrere Verbindungen getätigt werden, was über Ressourcen bzw. Ressouce Identifier abgebildet wird. Jede aufgebaute Verbindung wird mit der verwendeten JID und einem dahinter stehenden Ressource Identifier versehen, welcher beispielsweise die Form user@example.org/myPC haben kann und somit die Separierung für mehrere Geräte oder Verbindungen, die den selben Account verwenden, ermöglicht[8].

[5] Ein Client kann u.a. auch ein Bot oder andere Entität sein
[6] Siehe hierzu XEP-0124 und XEP-0206
[7] Wobei zu beachten ist, dass example.org dem Domainennamen des Servers entspricht.
[8] Siehe hierzu[17, S. 15]

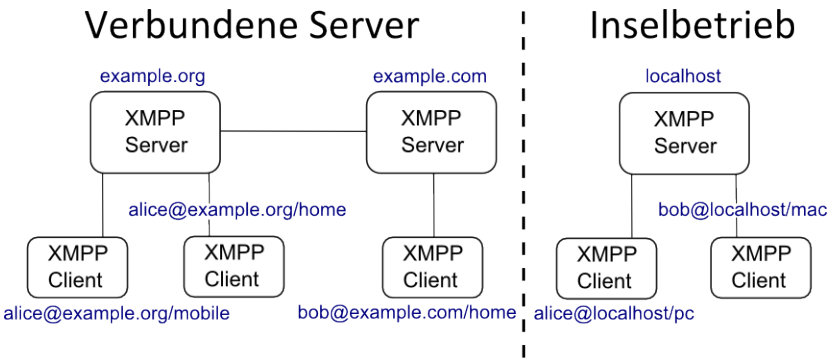

Abbildung 2: Beispielhafte XMPP-Architekturen angelehnt an [17, S. 14]

3.1.2 Kommunikation

Um die Dienste einer XMPP-Entität zu nutzen, muss als erster Schritt eine Verbindung aufgebaut werden. Für das Aufbauen der TCP-Verbindung wird die IP-Adresse der Entität, die die Nachricht empfangen soll[9] und der entsprechenden Port, auf dem gesendet werden soll, benötigt. Die IP-Adresse wird durch die Auflösung des Fully Qualified Domain Name (FQDN), sprich den absoluten Domain-Name des Empfängers bestimmt. Allerdings liegt oftmals nur die Domain selbst vor, welche auf mehrere Dienste verweisen kann. Aus diesem Grund muss der FQDN des Dienstes zuerst ermittelt werden. Dies wird bevorzugt durch einen DNS Service Lookup für die vorliegende Domain getätigt. Eine Beschreibung des DNS Service Lookup befindet sich für interessierte Leser in Abschnitt A.1. Der ermitteltet und ausgewählte FQDN wird dann zu einer IP-Adresse aufgelöst mit der die Entität eine TCP-Verbindung aufbauen kann.

Nach Aufbau der TCP-Verbindung wird der XML-Stream[10] eingeleitet, welcher im Fall einer Client-zu-Server-Verbindung nach dem folgenden Muster an den Server übertragen wird (siehe Algorithmus 1).
Die Nachricht beginnt mit einer optionalen XML-Deklaration, gefolgt von einem <stream>-Element, das den XML-Stream eröffnet. Das Prefix "stream" vor dem Doppelpunkt verweist auf den Namespace aus der 7. Zeile. Das <from>-Attribute beinhaltet die JID des Senders und das <to>-Attribute dessen Domainanteil. Da es sich um eine Client-zu-Server Kommunikation handelt, wird der Namespace auf "jabber:client" voreingestellt, so dass jedes Element diesen erhält, welches keinen Namespace angibt .

Der Server baut eine zweite Verbindung zum Client auf und sendet über diese ebenfalls

[9] Beispielsweise ein XMPP-Server, welche die Nachricht dann an einen Adressaten weiter schickt.
[10] Bei dem XML-Stream handelt es sich um einen Container für den Austausch von XML-Elemente, vgl. [16, Abschnitt 4.1].

Algorithmus 1 Aufbau des XML-Streams vom Client zum Server modifiziert aus [16, Abschnitt 4.2]

```
1 <?xml version='1.0'?>
2 <stream:stream
3         from='client@example.org'
4         to='example.org'
5         version='1.0'
6         xmlns='jabber:client'
7         xmlns:stream='http://etherx.jabber.org/streams'>
```

ein Stream-Element zum eröffnen des Streams (siehe Algorithmus 2). Dabei ist dieser identisch mit dem des Clients bis auf die ID, welche die Session eindeutig identifiziert. Im Anschluss daran werden von dem Server und dem Client weiter Verbindungsoptionen ausgehandelt, wie die Art der Authentifizierung oder Verschlüsselung, welcher hier nicht näher betrachtet werden. Der Client kann sich nun nach dem vom Server vorgegebenen Verfahren authentifizieren.

Algorithmus 2 Aufbau des XML-Streams vom Server zum Client modifiziert aus [16, Abschnitt 4.2]

```
1 <?xml version='1.0'?>
2 <stream:stream
3 from='example.org'
4         id='bu6fr48gkl'
5         to='client@example.org'
6         version='1.0'
7         xmlns='jabber:client'
8         xmlns:stream='http://etherx.jabber.org/streams'>
```

Nach diesen Schritten können nun asynchron, beliebig viele XML-Nachrichten ausgetauscht werden. Diese Nachrichten (oder auch Snippets in der englischsprachrigen Literatur) werden XML-Stanzas genannt. Es werden dabei die drei Arten Message, IQ und Presence von Stanzas unterschieden, welche jeweils einen Type haben, der je nach Art des Stanzas variiert und ein oder mehrere Kindelemente besitzen, die den Payload beinhalten.

Message Stanzas werden zu einem Empfänger gesendet, ohne dass eine Rückmeldung hinsichtlich der gesendeten Nachricht erforderlich ist[11]. Die möglichen Typen hier bei sind "normal" für einzelne Nachrichten ohne besondere Eigenschaft, "chat" für Echtzeitkommunikation zwischen zwei Entitäten, "groupchat" für Chaträume, "headline"

[11] Dieses Prinzip wird auch "fire and forget" genannt

für Alerts oder Benachrichtigung, wobei hier in keinem Fall eine Antwort gesendet werden soll und "error" für Fehlermeldungen.

Presence Stanzas zeigen die Verfügbarkeit von XMPP-Entitäten, wobei dies auch nach dem Publish-Subscribe-Paradigma funktioniert. Für die Vermittlung der Präsenzinformationen an andere Entitäten ist eine entsprechende Autorisierung durch die Entität erforderlich.

IQ (Info/Query-) Stanzas funktionieren nach den Request-Response Prinzip. Dabei gibt es 4 unterschiedliche Typen, wie "get" zum Abrufen, "set" zum Senden von Informationen, "result" als Antwort auf eine der vorherigen Anfragen oder "error" zum Übertragen einer Fehlermeldung.

Sobald der Informationsaustausch beendet werden soll, wird das Root-Element mittels eines </stream>-Elements geschlossen, was zur Trennung der TCP-Verbindung führt.

3.1.3 Notifications mit XMPP

XMPP bietet mit der XEP-0060 Spezifikation die Möglichkeit Nachrichten nach dem Publish-Subscribe-Paradigma zu verteilen. Dabei sind Nodes bei XMPP ähnlich zu Topics, die allerdings in zwei Typen unterschieden werden. Leafs sind Nodes auf denen Items (Notifications) veröffentlich werden können, während Collections Nodes sind, die keine Items sondern Leafs beinhalten und deren Beziehungen abbilden. Ein wichtiger Punkt dabei ist, dass Subscriber, die eine Collection abonniert haben, die Events aller Leafs der Collection erhalten.

Ein neuer Node kann entweder über einen Request oder die Auto-Creation angelegt werden. Die Auto-Creation tritt ein, wenn beim Service die entsprechende Option aktiviert wurde und eine Anfrage zum veröffentlichen einer Benachrichtigung zu einem Node gestellt wird, der nicht existiert. Der Service erstellt dann einen neuen Node, statt die Anfrage mit einer Fehlermeldung zurückzuweisen. Eine Anfrage zum Erstellen eines Nodes ist in Algorithmus 3 abgebildet, wobei zu beachten ist, dass der Publish-Subscribe Service im "to" Feld angesprochen wird und der Name des Nodes durch das Attritbut "node" im <create>-Element festgeleget wird. Zum Erstellen wird ein Stanza vom Typ IQ verwendet, damit der Sender eine Rückmeldung erhält, ob das Erstellen erfolgreich war oder eine Fehler aufgetreten ist.

Zusätzlich zum Erstellen von Nodes ist es möglich eine Vielzahl an Einstellungen für einen Node vorzunehmen, wie die persistente Speicherung von Items, wenn man eine History für Subscriber bereitstellen möchte. Ebenso kann entschieden werden, ob eine Notifikation einen Payload beinhaltet oder nicht, was dem Prinzip von Light und Fat Ping

aus Abschnitt 2.2 entspricht. Des Weiteren können auch Zugriffsbeschränkungen über eine Art Rollenmodell abgebildet werden. Darüber hinaus gibt es noch eine Reihe weiterer Einstellungen, welche Aufgrund ihres Umfanges hier nicht weiter ausgeführt werden.

Algorithmus 3 Erstellen eines Nodes modifiziert aus [14, Abschnitt 8.1]

```
<iq type='set'
    from='publisher@example.org/home-pc'
    to='pubsub.example.org'
    id='create1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <create node='TestNode'/>
  </pubsub>
</iq>
```

Damit eine XMPP-Entität bzw. Subscriber nun Benachrichtigungen erhält, muss die Entität den Node abonnieren. Hierfür schickt der Subscriber eine Anfrage an den XMPP-Publish-Subscribe Service, die Beispielhaft in Algorithmus 4 dargestellt ist. Die Anfrage darf nur ein <subscribe>-Element beinhalten, welche die JID enthalten sein muss. Da hier wieder ein IQ Stanzas verwendet wird, bestätigt der Service mittels einer Rückmeldung, ob die Subscription erfolgreich war.

Algorithmus 4 Abonnierung eines Nodes modifiziert aus [14, Abschnitt 6.1]

```
<iq type='set'
    from='subscriber@example.org/laptop'
    to='pubsub.example.org'
    id='sub1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
    <subscribe
      node='TestNode'
      jid='subscriber@example.org'/>
  </pubsub>
</iq>
```

In Algorithmus 5 ist das Veröffentlichen einer Nachricht dargestellt, wobei wieder ein IQ-Stanza verwendet wird. Das <publish>-Element sagt hierbei aus, dass der Sender eine Nachricht veröffentlichen möchte. Das <Item>-Element bezeichnet ein neues Event, in dem sich auch ein Payload befinden kann, welcher dann über ein <entry>-Element abgebildet wird. In diesem Fall ist kein Payload gegeben, sondern nur das Event[12]. Die Subsciber

[12] Ein Beispiel mit Payload wurde in Abschnitt A.2 hinterlegt.

enthalten darauf hin eine Nachricht, welche dem Schema aus Algorithmus 6 folgen. Hierbei werden Message-Stanzas verwendet, da der Publish-Subscribe Service nicht unbedingt Kenntnis darüber hat, ob der Subscriber online ist und welche Ressource genutzt werden soll, weshalb diese Entscheidung dem XMPP-Servers des Subscribers überlassen wird [17, S. 101-102].

Algorithmus 5 Veröffentlichen einer Nachricht ohne Payload, modifiziert aus [14, Abschnitt 7.1.1]

```
<iq type='set'
    from='publisher@example.org/home-pc'
    to='pubsub.example.org'
    id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
   <publish node='TestNode'>
     <item id='fwelp43gsdg4'/>
    </publish>
  </pubsub>
</iq>
```

Algorithmus 6 Nachricht die dem Subscriber zugestellt wird, modifiziert aus [14, Abschnitt 7.1.2.2]

```
<message from='pubsub.example.org'
         to='subscriber@example.org/laptop>
  <event xmlns='http://jabber.org/protocol/pubsub#event'>
    <items node='TestNode'>
      <item id='fwelp43gsdg4'/>
    </items>
  </event>
</message>
```

XMPP bietet über diesen Informationsaustausch hinaus auch die Möglichkeit Dienste von Entitäten, Nodes, Informationen und Metadaten von Nodes zu entdecken, damit es überhaupt zu einer Subscription kommen kann. Des Weiteren können Zugriffsberechtigungen, Items, Abonnenten zu Nodes abgerufen werden. Nähere Informationen zu dieser Thematik können [17, 14] entnommen werden.

3.2 Microblogging

Eines der Vorkommnisse des Web 2.0 ist das Microblogging, welches eine Kombination aus Kurzmitteilungen und Blogging darstellt. Neben Twitter, dem bekanntesten Vertreter der Microblogging-Dienste, sind eine Reihe ähnlicher Dienste entstanden, insbesondere solche, die die gleiche Funktionalität für Unternehmen anbieten[13]. Des Weiteren existieren für Twitter eine Vielfalt von Anwendungen und Diensten, die das Anwendungssprektrum erweitern oder die Nutzung vereinfachen[14]. Dies führt dazu, dass ein Teil der Anwender die Plattform nicht mehr aufruft, sondern die Funktionen über Anwendungen von Drittanbietern nutzt, womit sich Twitter mehr zu einer Kommunikationsmiddleware bewegt [1]. Da Twitter sowohl Echtzeitkommunikation als auch Anwendung-zu-Anwendung-Kommunikation mittels der vorgegebenen APIs ermöglicht, soll als nächstes betrachtet werden, wie eine verteilte Anwendung mit Twitter realisiert werden kann.

3.2.1 Twitter als Kommunikationsmiddleware

Um Twitter als Middleware und somit als schon funktionierende Infrastruktur für ein Echtzeitkommunikationssystem zu nutzen, müssen damit verbundene Restriktionen berücksichtig werden. Mit dem Statusupdate (oder auch Tweet) können Nachrichten nach dem Publish-Subscribe-Paradigma verteilt werden, bei denen die Follower (sprich die Subscriber) die Nachrichten erhalten. Allerdings sind Statusupdates auf 140 Zeichen begrenzt, so dass ähnlich zu den Verfahren aus Abschnitt 3.4 und 3.5 nur ein Light Ping-Verfahren möglich ist. Dies wird insbesondere deutlich, wenn die Verwendung von XML, JavaScript Object Notation (JSON), Really Simple Syndication (RSS) oder andere Auszeichnungssprachen in Betracht gezogen wird. Der Gedanke, der von Böhringer und Gluchowski [1] beschrieben wird, wobei nicht nur reale Personen sondern auch andere Entitäten als Informationsquelle dienen können, ist ein weiterer Grund zur Verwendung einer Auszeichnungssprache. Somit können Anwendung unabhängig voneinander diese Informationsquellen nutzen. Allerdings ergibt sich dabei die Problematik, dass bei der Verwendung der Timeline[15] für den Nachrichtenaustausch von Anwendungen, menschliche Follower durch diese Nachrichten gestört werden, womit ein zusätzlicher Account notwendig wäre.

Für Nachrichten bei denen mehr als ein paar wenige Parameter übertragen werden müssen, bietet es sich wegen der maximalen Zeichenlänge, dass Subscriber die Daten von dem Publisher mittels Pull abzurufen, was allerdings scheitern kann, wenn der Publisher nicht mehr erreichbar ist. Um diese Asynchronität zu gewährleisten wäre die Verwendung eines

[13] Beispielsweise https://www.yammer.com/ oder http://www.socialcast.com/
[14] Siehe hierzu Beispielsweise http://twtpoll.com/ und http://www.twhirl.org/
[15] Auf der Timeline werden die Tweets angezeigt

weiteren Dienstes oder eines eigenen Anwendungsserver erforderlich. Eine mögliche Architektur ist in Abbildung 3 dargestellt, wobei die Anwendungen via Twitter kommunizieren und einen Provider für den Austausch von Dateien oder längeren Nachrichten verwenden. Damit ein Subscriber antworten kann oder um eine Point-to-Point-Kommunikation zu initiieren, ist die Verwendung der Twitter Direktnachrichten möglich, welche allerdings auch auf 140 Zeichen begrenzt sind.

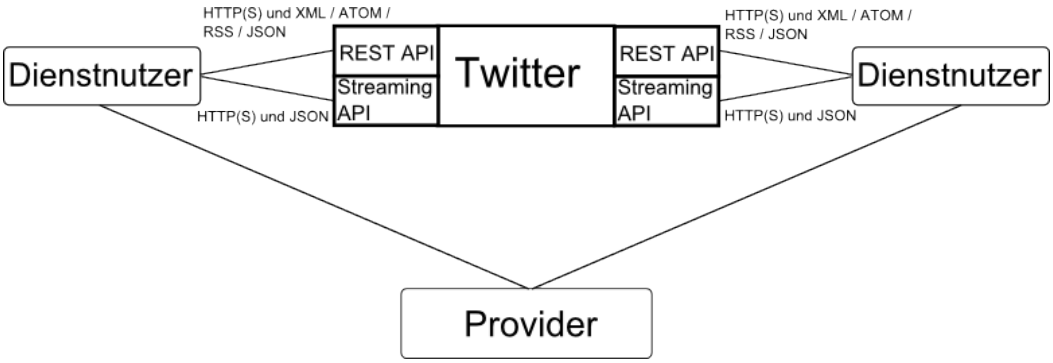

Abbildung 3: Eine mögliche Architektur mit Twitter als Kommunikationsmiddleware

3.2.2 Die Twitter-APIs und Authentifizierung

Damit nun die Kommunikation über Twitter stattfinden kann, sind Schnittstellen zur Nutzung der Plattform erforderlich. Twitter bietet für Entwickler eine REST API mit der die Kernfunktionalitäten wie Auslesen der Timeline und Nutzerinformationen oder das Erstellen von Statusupdates, sowie Retweets [22]. Des Weiteren können Informationen über die Streaming API in Echtzeit empfangen werden, was mittels einer langlebigen HTTP-Verbindung erreicht wird. Dabei werden die Datenformate XML, JSON, RSS und Atom zur Rückgabe, mit wenigen Ausnahmen, von der REST API unterstützt [23]. Die Streaming API dagegen gibt nur Antworten im JSON-Format [25]. Als letztes gilt es noch die Search API zu erwähnen, die eine Suchfunktion nach User ID, Geolocation, usw.[16] ermöglicht, hier aber nicht weiter behandelt wird.

Für die Umsetzung einer Anwendung-zu-Anwendung-Kommunikation bietet die REST-API synchrone Aufrufe nach dem Request-Response-Schema, wonach Direktnachrichten und Tweets verfasst werden können. Das Abrufen der Nachrichten in Echtzeit wird somit allerdings nicht gewährleistet, weshalb hier auf die Streaming API zurück gegriffen werden muss. Dabei erfolgt die Authentifizierung über ein spezifisches OAauth Verfahren, welches

[16] Das Ergebnis können dabei die Tweets zu einem Ort sein oder die Tweets eines Twitternutzers.

das ältere Verfahren mit Nutzernamen und Passwort (auch HTTP Basic Authentification genannt) fast vollständig ablöst hat.

Das OAuth Verfahren adressiert das Problem, dass Anwender die eine Plattform über Drittanbieter nutzen wollen, ihre Logindaten preisgeben müssen. In so einem Fall hat der Drittanbieter uneingeschränkten Zugriff auf alle Ressourcen des Anwenders [7]. Des Weiteren muss der Nutzer nach der Nutzung des Drittanbieters das Passwort ändern, um sicher zu gehen, dass der Drittanbieter diesen Zugang nicht mehr nutzt. Das hat zur Folge, dass auch alle anderen Anwendungen die dieses Passwort benutzen, dass neue Passwort benötigen. Zusätzlich ist es möglich, dass der Nutzer die gleichen Zugangsdaten auch auf anderen Plattformen, wie Online-Banking, Social-Networks, E-Mail Account usw. verwendet, was ein weiteres Risiko darstellt. Beim OAuth Verfahren erlaubt der Besitzer der Ressourcen (Fotos, Dateien, etc.) auf einem Dienstanbieter den Zugriff für einen Client mit einem eigenen Zugang. Somit kann dieser Zugang bei Bedarf gesperrt werden, ohne das Passwort zu ändern. Des Weiteren kann der Zugang in Dauer, Umfang etc. beschränkt werden. Eine Beschreibung zum OAuth Verfahren, welches in dem RFC5849[17] spezifiert ist, kann aus [8] entnommen werden.

Die Streaming API welche nur über HTTPS mit SSL angesprochen werden kann, wird dabei in drei Teile unterschieden, die Streaming API, User Streams und Site Streams. Bei der Streaming API, welche als einzige noch über das HTTP Basic Authentification Verfahren genutzt werden kann, können die Timelines aller öffentlicher Profile angezeigt werden [24]. User Streams erweitern dies um Nutzerspezifische Informationen, wie Direktnachrichten und können nur über das OAuth Verfahren genutzt werden. Bei einer gewissen Anzahl von mehreren User Streams über einen Host müssen Site Streams verwendet werden. Der Verbindungsaufbau erfolgt bei der Streaming API mittels eines HTTP Request, wobei der Twitter Server die Verbindung dann aufrecht erhält, um Nachrichten an den Client zu versenden.

3.2.3 Mögliche Kommunikationsabläufe

Ein mögliches Anwendungsszenario unter der Verwendung der Twitter APIs könnte also folgendermaßen aussehen. Dienstnutzer veröffentlichen Nachrichten mittels der REST API auf ihrer Timeline. Dabei können dies Informationen gesteuert durch einen Nutzer, automatische Nachrichten einer Software auf dem Endgerät eines Anwenders oder größere Informationsdienste, wie Wetterdienst, Kino, Börsen, Firmen, Katastrophenschutz, etc. sein. Die Dienstnutzer können in Echtzeit über die Streaming API diesen Diensten folgen, um die Informationen zu nutzen. Die Direktnachrichten bieten sich an, um entsprechend

[17] siehe http://tools.ietf.org/html/rfc5849

Antworten zu können oder eine Point-to-Point Kommunikation zu initiieren. In einigen Fällen sollte ein weiteres System verwendet werden, damit größere Nachrichten vermittelt werden können. Alternativ ist es möglich Twitter ergänzend für die Architektur einer Anwendung zu verwenden, um Echtzeitbenachrichtigungen zu ermöglichen. Ein weiterer wichtiger Punkt bei einem solchen Systemdesign ist sicherlich eine einheitliche Schnittstelle zu externen Systemen, damit jeder mit jeden kommunizieren kann und nicht nur ein Anwendungstyp unter sich. Des Weiteren muss eine Auszeichnungssprache und Schema verwendet werden, so dass Anwendungen unabhängig von der Implementierung mit einander kommunizieren können. Hinsichtlich der Restriktionen von Twitter entseht momentan auch ein weiteres Angebot unter http://www.long-tweets.com, welches es ermöglicht Bilder und mehr als 140 Zeichen zu versenden.

3.3 Amazon SQS und Amazon SNS

Amazon bietet seit 2006 mit dem Amazon Web Service (AWS) die Möglichkeit verschiedene Webdienste oder auch Rechner- und Speicherkapazität zu mieten. Dies ermöglicht den Kunden den Aufbau einer eigenen IT-Infrastruktur zu umgehen und die benötigte Dienstleistung ohne großen Aufwand zu skalieren. Zu diesen Diensten gehören u.a. der nachrichtenorientierte Kommunikationsdienst Amazon Simple Queue Service (SQS), welche das Senden und Empfangen von Nachrichten mittels Warteschlangen ermöglicht und der Amazon Simple Notification Service (SNS), welcher die Nachrichtenverteilung nach dem Publish-Subscribe Paradigma bereitstellt. Beide Dienste sind bis zu einem gewissen Datenvolumen oder Anfragekontingent kostenlos, wobei allerdings bei der Erstellung eines AWS Accounts eine Kreditkarte angegeben werden muss. In der Nutzung von Amazon SNS sind Nachricht von Amazon SQS enthalten, weshalb SQS Nachrichten hier kostenlos sind. Im weiteren wird die Funktionsweise dieser beiden Verfahren beschrieben.

3.3.1 Kommunikation mit Amazon SQS

Bei Amazon SQS können beliebige viele Sender (Producer) und Empfänger (Consumer) zu einer Warteschlange (oder auch Queue) existieren. Es ist allerdings auch möglich dies einzuschränken, indem bestimmt wird, welcher Account Nachrichten an die Warteschlange senden und welcher Nachrichten von der Warteschlange abrufen oder entfernen darf. Die Warteschlangen befinden sich auf den Server von Amazon und können über eine REST Web Service Schnittstelle über HTTP oder HTTPS angesprochenen werden (siehe Grafik 4). Es können dabei auch Optionen wie Erstellen, Bearbeiten und Löschen von Warteschlangen durchgeführt werden. Durch die Warteschlange werden Sender und Empfänger entkoppelt und somit eine asynchron Kommunikation gewährleistet. Allerdings hat das Abrufen der

Nachrichten über die REST Schnittstelle zur Folge, dass nur über Pull die Daten beim Dienstnutzer aktuell gehalten werden können. Beim Abrufen wird nicht garantiert, dass die Reihenfolge der Nachrichten erhalten bleibt, wie sie an die Warteschlange gesendet wurden.

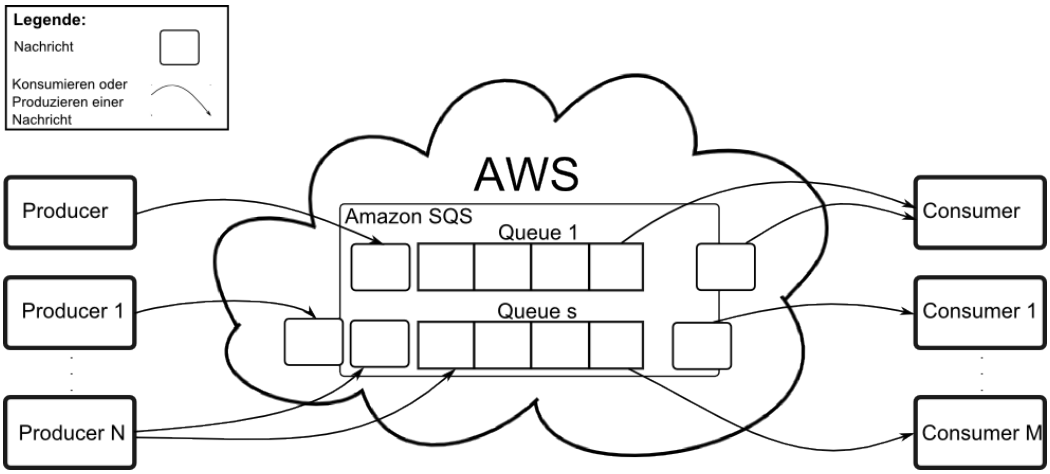

Abbildung 4: Schematische Abbildung der Informationsverteilung mittels Amazon SQS

Für einen Kommunikationsablauf, wobei die Warteschlangen schon entsprechende eingerichtet wurden, wird eine Nachricht mittels eines HTTP(S) Get- oder Post-Request übertragen [18, S. 21-22] (siehe Listing 7). Der Parameter Action ist bei jeder Anfrage zwingend erforderlich und bestimmt welche Aktion ausgeführt werden soll[18]. Abhängig von dem Parameter Action variieren die weiteren Paramater, wobei einige gleich bleiben. Im MessageBody befindet sich der Inhalt der Nachricht, welcher als einziger Parameter Leerzeichen enthalten darf und auf 64 KB begrenzt ist. Der Parameter Version identifiziert die API-Version, die verwendet werden soll. Das Datum für Expires steuert, dass Nachrichten, die erst nach einem gewissen Zeitraum eintreffen, abgelehnt werden.

AWS akzeptiert nur authentifizierte Anfragen, weshalb die Version der Signatur und Verschlüsselungsmethode, die AWSAccesKeyID und eine Signatur in den Anfragen enthalten sein muss. Die AWSAccessKeyID kann über das AWS Identity and Access Management (IAM) bezogen werden, wobei Berechtigungen für Nutzer eingerichtet oder auch Nutzer und Nutzergruppen erstellt werden können. Zu dem AWSAccessKey existiert zusätzlich ein Secret Access Key, welcher zur Verschlüsselung genutzt wird und nur dem Absender des Request und dem AWS bekannt ist. Die Signatur wird mittels der HMAC-SHA Hash-Funktion zusammen mit dem Inhalt des Requests und dem geheimen Schlüssel berechnet

[18] "SendMessage" steht hier beispielsweise für das Senden einer Nachricht

Algorithmus 7 Ein HTTP Get-Request zum Erstellen einer Nachricht bei Amazon SQS aus[18, S. 21]

```
1 http://sqs.us-east-1.amazonaws.com/123456789012/queue1
2 ?Action=SendMessage
3 &MessageBody=Your%20Message%20Text
4 &AWSAccessKeyId=AKIAIOSFODNN7EXAMPLE
5 &Version=2011-10-01
6 &Expires=2011-10-15T12:00:00Z
7 &Signature=lBP67vCvGlDMBQ1dofZxg8E8SUEXAMPLE
8 &SignatureVersion=2
9 &SignatureMethod=HmacSHA256
```

(siehe Abbildung 5). Die Nachricht muss dann nur von dem AWS mittels der gleichen Berechnung verifiziert werden, wobei zum einen die Integrität der Nachricht und die Authentizität des Senders sichergestellt wird, da dieser der einzige ist, der diese Signatur erstellen kann, solange niemand anderes über den geheimen Schlüssel verfügt. Der AWS kann mittels des gleichen Weges die Echtheit kontrollieren, wie in Abbildung 6 dargestellt ist. Das gleiche Verfahren kann auch für das Abrufen von Nachrichten, zum Erstellen von Warteschlangen etc. genutzt werden. Eine Liste und Beschreibung der möglichen Verfahren wird in [19] dargelegt.

3.3.2 Kommunikation mit Amazon SNS

Bei Amazon SNS können Topics erstellt werden, welche dann zur Nachrichtenverteilung nach dem Push-Verfahren an Abonnenten genutzt werden können. Zum Abonnieren müssen Endpunkte angegeben werden, wobei diese eine Warteschlange, E-Mailadresse, Telefonunummer oder URL eines Web Servers sein können [20, S. 32]. Dabei werden die Protokolle SMTP (E-Mail), HTTP, HTTPS, und SMS unterstützt. Dies bedeutet, dass nur ein Teil der Subscriber mittels Push-Verfahren Nachrichten zugestellt bekommen, da zwar eine Nachricht an eine Warteschlange via Push zugestellt wird, der Empfänger die Warteschlange aber trotzdem regelmäßig abfragen muss. Teils ist ein Push-Benarchitgung zum Endgerät also nur über Verwendung weiterer Technologien bzw. Komponenten möglich.

Zur Interaktion wie Senden und Empfangen steht wie schon bei Amazon SQS eine REST Schnittstelle zur Verfügung. Die Abläufe sind mit denen von Amazon SQS vergleichbar und bringen keine neuen Erkenntnisse, weshalb diese hier nicht näher behandelt werden. Da mehrere Subscriber ein Topic abonnieren und dabei ein unterschiedliches Protokoll verwenden können, existiert die Möglichkeit bei Amazon SNS verschiedene Nachrichtentext

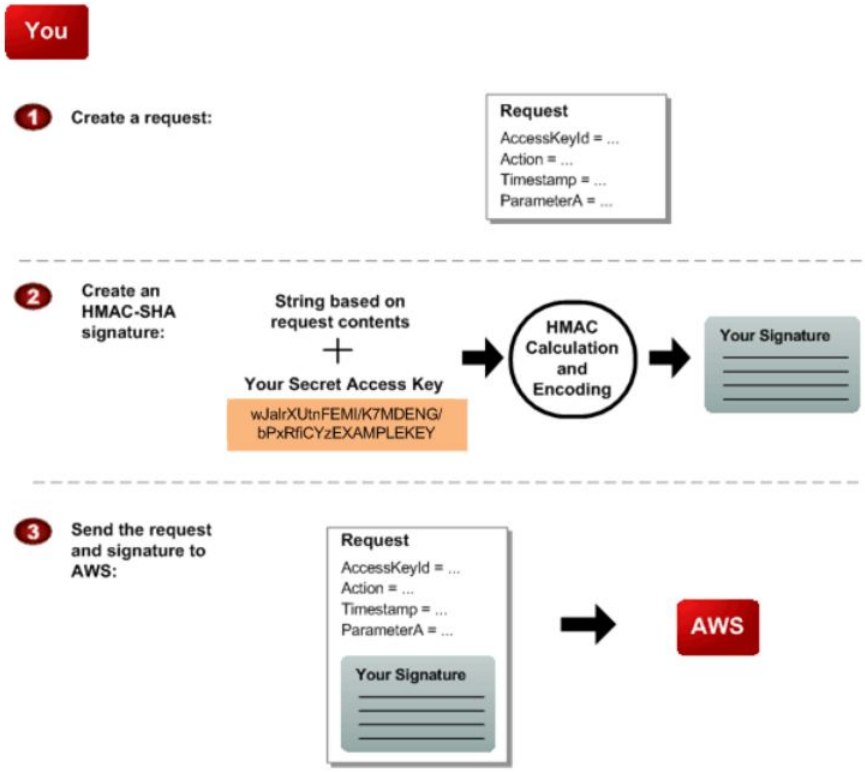

Abbildung 5: Authentifizierungsprozess zum AWS hin aus [18, S. 28]

in einer Nachricht für jedes Protokoll zu definieren. Dies ist erweist sich als sinnvoll, wenn man Restriktionen wie die begrenzte Zeichenanzahl bei SMS betrachtet. Des Weiteren kann mit Amazon SNS der Ersteller eines Topics verschiedene Berechtigungen für den Zugriff auf ein Topic vergeben.

3.4 Apple - Push Notification Service

Apple bietet mit dem Push Notification Service die Möglichkeit den Nutzer eines iPod, iPad, iPhones oder Macintosh zu informieren, dass eine Anwendung über neue Informationen verfügt bzw. ein neues Ereignis eingetreten ist, solange die Anwendung sich nicht im Vordergrund befindet[19] [9, S. 20]. Die Benachrichtigung wird über das Betriebssystem als Ton, Alert Message oder Badge Number repräsentiert[20]. Der Nutzer kann daraufhin, nach eigenen Ermessen die Anwendung aufrufen, um weitere Aktionen durchzuführen.

[19] Anwendungen, die schon laufen, können diese auch direkt verabreiten.
[20] Siehe Abbildung 7

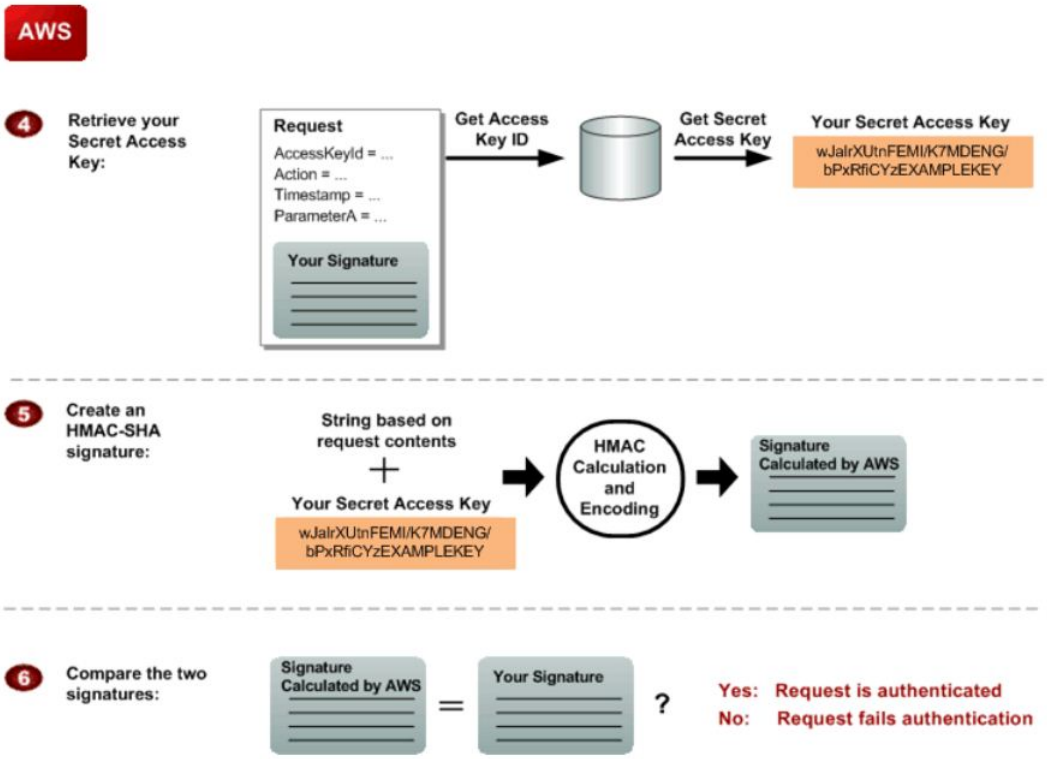

Abbildung 6: Authentifizierierungsprozss des AWS aus [18, S. 29]

Die Benachrichtigung wird dabei über einen Drittanbieter (oder auch Provider) mittels einer Push Notification oder von einer Anwendung über eine Local Notification ausgelöst. Beide sind von der Art der Benachrichtigung gleich. Local Notfications sind auf dem iOS 4.0 verfügbar und geben den Auftrag für eine Benachrichtigung, die dann von dem Betriebssystem entsprechend ausgeführt wird[21]. Push Notifications dagegen werden von außerhalb über den Apple Push Notification Service (APNs) durch den Provider ausgelöst und sind seit iOS 3.0 und Mac OS X 7.0 in die Betriebssysteme von Apple integriert. Der Nutzdatenanteil der Nachrichten, die über den APNs verschickt werden können, ist auf 256 Bytes begrenzt und bestimmt, wie der Nutzer informiert werden soll. Wird das Limit überschritten, weist der APNs die Anfrage ab. Da die Benachrichtigung nur auf Betriebssystemebene erfolgt, handelt es sich hier nicht um eine Anwendung-zu-Anwendung-Kommunikation. Dies heißt also auch, dass die Anwendung erst Informationen von dem Provider beziehen kann, wenn sie gestartet wird und einen weiteren Kommunikationskanal

[21] Ein Anwendungfall hierfür könnte eine Eieruhr sein, die nach einer vorgegebenen, verstrichenden Zeit den Nutzer informiert

Abbildung 7: Eine Alert Message (links) und Badge Number (rechts) aus [9, S. 12 - 13]

unabhängig von dem APNs aufgebaut.

> *"Once alerted, users may choose to launch the application, which then downloads the data from its provider."* [9, S. 14]

Der Vorteil des Push Notification Service liegt in der dauerhaften, verschlüsselten IP-Verbindung, die zwischen den Geräten und dem APNs besteht, wenn dieser Dienst aktiviert ist. Somit können eine Vielzahl von Drittanbietern Push-Benachrichtigungen versenden, ohne das jeder eine eigene Verbindung benötigt bzw. ständiges Polling der Anwendungen durchgeführt werden muss.

3.4.1 Kommunikationsablauf und Nachrichtenstruktur

Um eine Kommunikation für eine Anwendung einzurichten, muss die Anwendung sich mit dem Betriebssystem auf einem Gerät beim APNs registrieren. Hierfür bezieht die Anwendung einen Device Token über das Betriebssystem, welcher von dem APNs generiert wird. Danach reicht die Anwendung den Device Token an seinen Provider weiter (siehe Abbildung 8). Die Anwendung ist für die Übermittelung des Tokens an den Provider selbst zuständig und muss hierfür einen anderen Übertragungsweg wählen als über den APNs[22].

Mittels des Token ist es für den Provider möglich, Nachrichten an das Gerät über den APNs zu senden. Das Anwendergerät muss zum Empfangen von Nachrichten eine dauerhafte Verbindung zum APNs aufbauen, welcher die Nachrichten des Providers an das

[22] Dies gilt für alle Nachrichten, die von dem Gerät an den Provider gesendet werden sollen

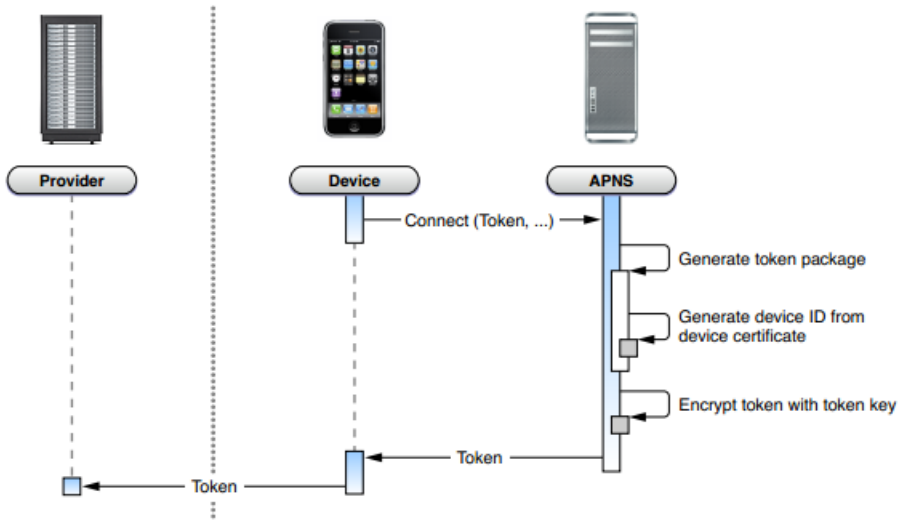

Abbildung 8: Beziehen eines Device Token aus [9, S. 31]

Gerät ausliefert. Da immer nur ein Gerät mittels mittels des Tokens adressiert werden kann, handelt es sich hierbei um eine Point-to-Point-Kommunikation (siehe Abbildung 9). Der Token kann sich beispielsweise durch das Ausführen der Time Machine durch den Benutzer ändern, womit es erforderlich ist, dass Registrieren und Weitergeben des Tokens regelmäßig zu wiederholen.

Für den Fall, dass der APNs eine Nachricht nicht zustellen konnte, wird diese für eine begrenzte Dauer gespeichert und zugestellt, sobald das Gerät wieder verfügbar ist. Dennoch kann es dazu kommen, dass der APNs die Nachricht nicht zustellen kann, da Beispielsweise die Anwendung von dem Anwender entfernt wurde. Hierfür sollte der Provider den Feedback-Service von Apple verwenden und in den Fällen, bei denen sich die Anwendung nicht mehr erneut registriert, das Senden weiterer Benachrichtigungen an das Gerät einstellen. Zur Kommunikation mit dem Feedback-Service wird ein binäres Interface für Push-Benachrichtigungen mittels TLS oder SSL verwendet.

Damit der Provider Nachrichten an den APNs senden kann, steht ein binäres Interface, welches über ein asynchrones TCP-Socket erreichbar ist, zur Verfügung. Das Interface sieht dabei einen erweiterten oder simplen Nachrichttypen vor. Der simple Nachrichtentyp umfasst dabei ein Command Byte mit dem Wert 0, die Länge des Payloads und des Device Token, sowie dem Device Token und den Payload selbst (siehe Abbildung 10). Der Payload besteht aus einem JSON Dictonary.

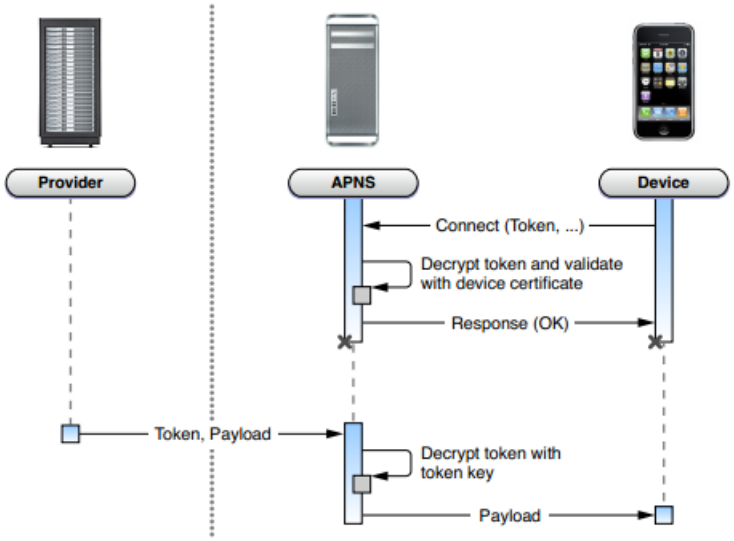

Abbildung 9: Kommunikation über den APNs aus [9, S. 32]

Abbildung 10: Simple notification format aus [9, S. 44]

3.5 Android - Cloud 2 Device Messaging

Ähnlich zu dem Push Notfication Service von Apple, gibt es für Geräte mit Android das Google Cloud 2 Device Messaging Framework (C2DM). Über C2DM können Provider Push Benachrichtigungen an ihre Anwendungen verschicken, wobei eine stetige Verbindung verwendet wird. Diese wird auch für andere Google Services, wie Beispielsweise G-Mail, Google Calendar oder dem Play Store[23]. Aus diesem Grund ist ein Google Account für den Anwender verpflichtend, wenn dieser C2DM auf dem Gerät nutzen möchte. Des Weiteren wird Android 2.2 oder eine höhere Version, sowie ein installierter Play Store auf dem Gerät vorausgesetzt[24]. Der Nutzanteil der Daten für eine Nachricht ist auf 1024 Bytes begrenzt und Nachrichten können nur in eine Richtung versendet werden, was bedeutet dass hier

[23] ehemals Android Market
[24] Dies heißt allerdings nicht, dass die Anwendung im Market verfügbar sein muss.

wieder anderweitige Kommunikationswege verwendet werden müssen, möchte man größere Datenmengen an das Gerät oder Daten von dem Gerät an den Provider senden.

Im Vergleich zum APNs bietet C2DM eine Anwendung-zu-Anwendung-Kommunikation, da die Daten ohne Nutzerinteraktion bis zur Anwendung weitergeben werden. Der Entwickler hat hierbei die Möglichkeit selbst zu bestimmen, den Anwender über das Ereignis zu informieren oder einen Prozess im Hintergrund auszuführen, der ohne die Interaktion des Anwenders ausgeführt werden kann. Der Anwender muss allerdings den entsprechend Berechtigungen beim ersten Start der Anwendung zustimmen.

3.5.1 Kommunikationsablauf

Die Übertragung mittels dem C2DM Framework erfordert verschiedene Schritte, wie Registrierung, Authentifizierung und Senden, sowie Behandeln der Nachrichten. Diese Schritte sind in Abbildung 11 dargestellt und werden im weiteren Verlauf beschrieben.

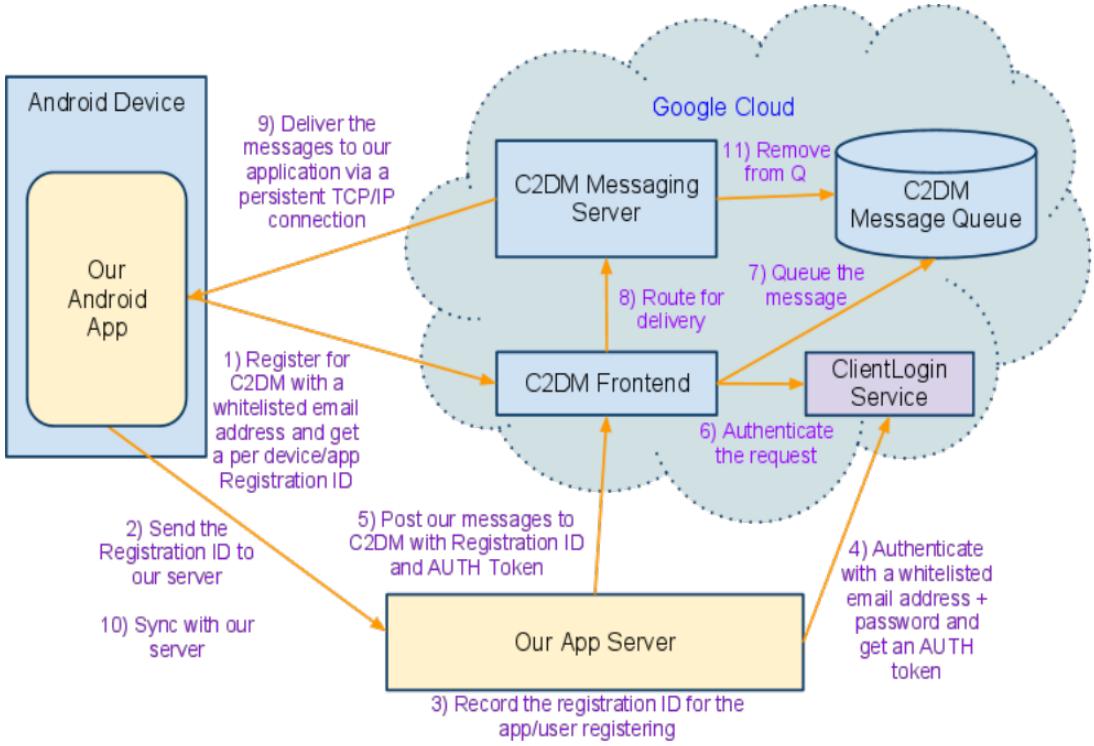

Abbildung 11: Kommunikationsabläufe bei C2DM aus [4, S. 14]

Der erste Schritt um C2DM für eine Anwendung nutzen zu können, ist die Anmeldung bei Google mit einer eigenen E-Mailadresse, da sich C2DM noch in der Testphase

befindet[25]. Der Zugriff auf C2DM wird erst nach eine Überprüfung von Google freigegeben und kann eine gewisse Zeit dauern[26]. Die freigeschaltete E-Mailadresse (auch Sender ID genannt) wird sowohl von den Androidapplikationen und dem Provider benötigt, um sicherzustellen, dass sie die Berechtigung zur Nutzung des C2DM Frameworks besitzen. Dabei muss die E-Mailadresse nicht personenbezogen sein, sondern kann auch den Namen der Anwendung oder ähnliches tragen.

Damit es nun zur Übertragung von Nachrichten kommen kann, muss die Anwendung sich beim C2DM Server registrieren und sendet dafür lokal auf dem Gerät einen Intent zur Registrierung mit der Sender ID und der Application ID. Die Application ID wird durch die Paketnamen im Manifest spezifiziert. Das Betriebssystem sendet die Anfrage an den C2DM Server weiter, welcher bei erfolgreicher Registrierung die Registration ID generiert und zurück gibt[27]. Diese wird an die Anwendung über das Betriebssystem mittels eines Intent weitergeben, welche wiederum auf einem eigenen Übertragungsweg, die Registration ID an den Provider schickt (siehe Punkt 2 in Abbildung 11).

Der Provider legt die Registration ID im Regelfall persistent ab und verwendet diese zusammen mit einem Auth Token zum versenden von Nachrichten an das Gerät über den C2DM Server. Der Auth Token dient zur Sicherstellung, dass der Provider für die Anwendungen eines Typs, Nachrichten über den C2DM Server versenden darf. Der Entwickler sollte den Auth Token möglichst schon vorher von dem Android Framework bezogen haben. Hierfür muss ein HTTPS Post Request durchgeführt werden, welcher u.a. die Sender ID, ein Passwort und den Servicetypen[28] enthält (siehe Punkt 4 in Abbildung 11). Eine genaue Beschreibung hierfür kann aus [12] entnommen werden.

Die Nachrichten des Providers adressieren die Geräte mittels der die Registration ID, welche eindeutig einem Gerät zugeordnet ist, womit nur eine Point-to-Point-Kommunikation möglich ist. Die Nachrichten können vom Provider an den C2DM Server mittels eines HTTP-Post Request gesendet werden, bei dem die Registration ID, der Nutzdatenanteil, der Auth Token, eventuelle Verzögerungseinstellungen[29] und einen Collapse Key[30] mit übertragen werden. Der Payload setzt sich auch einer unbegrenzten Anzahl von Key-Value Pairs zusammen[31]. Sollten hierbei keine Fehler auftreten[32], da beispielsweise der Service

[25] siehe http://code.google.com/intl/de-DE/android/c2dm/signup.html

[26] Die Freischaltung einer E-Mailadresse für ein Proof-of-concept für diese Ausarbeitung dauerte 1,5 Tage.

[27] Die Registrierung kann aus verschiedenen Gründen fehlschlagen, worauf die Anwendung entsprechend reagieren sollte [11].

[28] Hier wird der Name des zu verwendenen Dienstes angegeben, wie ac2dm für Cloud 2 Device Messaging

[29] Hierbei kann man angeben, in dem man den Parameter auf true setzt, dass die Nachricht erst übertragen wird, wenn der Bildschirm des Gerätes nicht gesperrt ist.

[30] Über den Collapse Key können Nachrichten einer Gruppen zugeordnet werden.

[31] Die Begrenzung erfolgt durch die maximale Größe der Nachricht

[32] Fehler werden als Rückgabewert des Request vermittelt und können aus [11] entnommen werden.

nicht erreichbar oder der Auth Token ungültig sein kann, wird die Nachricht über eine persistent TCP/IP Verbindung mit SSL-Verschlüsselung an das Gerät gesendet [4, S. 15] (siehe Punkt 5 und 9 und in Abbildung 11). Dafür werden innerhalb der Google Cloud mehrere Schritte durchlaufen, wie die Überprüfung des Auth-Tokens, das Weiterleiten der Nachrichten zu einem Server, der die persistente Verbindung zum dem Gerät hat oder das Ablegen der Nachricht in eine Queue, wenn die Nachricht nicht direkt übermittelt werden kann [5, Min. 58] (siehe Punkte 6 bis 11 in Abbildung 11). Im letzten Fall bleibt die Nachrichten in der Queue erhalten, bis das Gerät wieder verfügbar ist. Dabei wird allerdings immer die letzte Nachricht mit dem gleichen Collapse Key und Registration ID überschrieben, sprich es wird immer nur eine Nachricht pro Collapse Key gespeichert [11]. Dies soll verhindern, dass eine Anwendung viele Benachrichtigungen für einen Dienst erhält, da es ausreichend ist, der Anwendung einmal mitzuteilen, dass neue Daten beim Provider vorliegen. Des Weiteren wird nicht garantiert, dass die Reihenfolge der Nachrichten erhalten bleibt. Die letzte Nachricht des Providers muss also nicht die letzte Nachricht sein, die vom C2DM Server versendet werden [11]. Somit sollte der aktuelle Stand beim Provider gespeichert werden.

Das Android-Betriebssystem empfängt die Nachricht von dem C2DM Server[33] und sendet die Key-Value Pairs mittels eines Broadcast Intent an die Anwendung weiter. Die Anwendung wird als Adressat für den Intend anhand entsprechender Angaben in der Manifest.xml-Datei ausgemacht und durch den Intent aufgeweckt[34]. Die Key-Value Pairs können dann von der Anwendung aus dem Intent extrahiert und weitere Aktion können durchgeführt werden, wie das Abrufen von aktuellen Daten von dem Provider über einen eigenen Übertragungsweg. Zum Senden weiterer Benachrichtigungen an ein Gerät sind nur die Schritte 5-11 aus Abbildung 11 erforderlich.

[33] Dies wird über einen Hintergrundprozess (Service) realisiert, der gestartet wird und sobald eine Netzverbindung besteht und die persistente Verbindung zum C2DM Server aufbaut [4, S. 15].

[34] D.h die Anwendung muss voher nicht ausgeführt werden [11]

4 Kategorisierung und Dimensionen

Durch die Betrachtung der verschiedenen webbasierter Technologien für Echtzeitkommunikation, konnten einige Dimensionen identizifiziert werden, die sich dazu eignen Architekturen in einem Anwendungskontext zu bewerten oder Ansatzpunkte für eigene Architekturlösungen zu entdecken. Da allerdings der Fokus dieser Ausarbeitung auf der Funktionsweise liegt, wurden die Dimensionen hauptsächlich aus dieser Perspektive identifziert.

Alle identifziertien Dimensionen wurden in Kategorien unterteilt, wobei der Architekturaufbau in Abschnitt 4.1 behandelt wird. Die Eigenschaften der Übertragung gehören zu einem Teil auch zu dem Architekturaufbau, wurden hier allerdings gesondert in Abschnitt 4.2 behandelt. Dimensionen, welche für die Sicherheitsaspekte der Übertragung identifiziert werden konnten, wurden unter Sicherheit in Abschnitt 4.3 dargelegt. Alle Eigenschaften, die nicht diesen Kategorien zugeordnet werden konnten, wurden unter Sonstiges in Abschnitt 4.4 vermerkt.

4.1 Architekturaufbau

Bevor die hier behandelten Technologien im Detail untersucht wurden, gab es eine Recherchephase, in der ein breites Spektrum von möglichen Lösungen für asynchrone Echtzeitsysteme betrachtet. Dabei stellt sich heraus, dass die Technologien unterschiedlich klassifiziert werden können, wie etwa nach Service, Framework, Architekturstil, Standard, Protokoll, Middleware usw. Die Klassifzierung der Technologien wurde in Tabelle 1 festgehalten, wobei das C2DM und der APNS als Service und Framework gezählt werden können, da sie zum einen für den Provider als Service im Web zur Verfügung stehen, allerdings auf den Geräten wie ein Framework zu verwenden sind. Die zweite Dimension ist der zentrale bzw. dezentrale Aufbau einer Technologie, deren Bedeutung schon in Abschnitt 2.3 herausgestellt wurde. Konzeptionell lassen sich einige Vor- und Nachteil ableiten, die sich durch den Aufbau der Architekur ergeben. Mögliche Schwächen von zentralen Technologien sind u.a. die große Lasten für Speicher, Bandbreite und Berechnungen, die an einer Komponente auftreten [13]. Vergleicht man dezentrale Client-Server Architekturen mit reinen Peer-to-Peer Architekturen, so sind diese leichter zu verwalten und weniger anfällig für Ausfälle [17, S. 11]. Zentrale Architekturen dagegen können Vorteile haben, wie eine konsistente Datenhaltung oder auch eine einfache Verwaltung, sowie ein Maß an Kontrolle.

4.2 Dimensionen der Nachrichtenübertragung

Die Dimensionen der Nachrichtenübertragung wurden zur Übersicht in die zwei Tabellen 2 und 3 unterteilt. Die Dimension Anwendung-zu-Anwendung Kommunikation gibt Auf-

	Twitter	XMPP	Amazon SNS	Amazon SQS	Android C2DM	Apple Push Notification
Klassifikation	Service	Offener Standard	Service	Service	Framework Service	Framework Service
Dezentral/ Zentral	zentral	dezentral	zentral	zentral	zentral	zentral

Tabelle 1: Einteilung der Technologien nach Architekturaufbau

schluss darüber, ob Anwendungen miteinander kommunizieren können oder ob es sich um eine Anwendung-zu-Mensch bzw. Mensch-zu-Mensch Kommunikation handelt. APNs wurde als einzige Technologie nicht als Anwendung-zu-Anwendung Kommunikation gezählt, da die Interaktion des Nutzers oftmals erforderlich ist.

Das Kommunikationsparadigma bzw. -möglichkeiten zeigen welche Arten der Kommunikation durch die Technologien ermöglicht werden. XMPP bietet dabei ein breites Spektrum an Möglichkeiten. Da bei Amazon SQS mehrere Empfänger und Sender über eine Warteschlange kommunizieren können ebenfalls mehr Möglichkeiten als die reine Point-to-Point Kommunikation. Die Kommunikationsrichtigung wurde als ein weiteres Merkmal identifiziert, da beispielsweise C2DM und APNs nur den Übertragungsweg vom Provider zum Endgerät abdecken. Dies hat zur Folge, dass weitere Kommunikationswege geschaffen werden müssen, wenn eine die Endgeräte auch mit dem Provider kommunizieren sollen. Damit in Verbindung steht auch die Dimension Kommunikationsstil bei der festgelegt ist, welche der Kommunikationsteilnehmer direkt miteinander kommunizieren können.

Die Nachrichtenbegrenzung und die Unterscheidung zwischen Fat und Light Ping sind miteinander verknüpft. Auch wenn es möglich ist mit Technologien wie APNS und C2DM einen kleinen Nutzdatenanteil zu versenden, kann man diese nicht unbedingt als Fat Ping einordnen, da der Anteil sehr gering ist. Die Einordnung zeigt, dass diese Unterscheidung nicht komplett binär getroffen werden kann, da einige Verfahren Einschränkungen im Bezug auf den Nutzdatenanteil besitzen, aber trotzdem ausreichend um einen großen Bereich von Anwendungsszenarien abzudecken, wie es Beispielsweise bei Amazon SQS der Fall ist. Wie schon in Abschnitt 2.2 beschrieben, können sich Vor- und Nachteile durch die Verwendung von Fat und Light Ping ergeben.

Die letzten drei Dimensionen betreffen das Transportprotokoll, die Nachrichtenformate und ob Nachrichten nach dem Pull- oder Push-Verfahren vermittelt werden. Verfahren die nach der Pull-Variante arbeiten, verwenden hier meist synchrone HTTP(S)-Requests, während bei der Push-Variante oftmals persistente TCP-Verbindung oder langlebige HTTP(S)-

Verbindungen realisiert verwendet werden. Ausnahmen sind dabei SMS und SMTP im Fall von Amazon SNS. Bei Verfahren, welche auf Pull basieren, sollte beachtet werden, dass es durch den wiederholten Aufbau einer TCP-Verbindung zu einem erhöhten Overhead, mehr CPU Zyklen und Bedarf an Cache Speicher kommt[35]. Dies kann sich auch auf mobilen Geräten bemerktbar machen, weshalb hier Technologien wie C2DM und APNS von Vorteil sind. Die Nachrichtenformate sind je nach dem Anwendungsszenario gesondert zu betrachten, weshalb hier keine weiteren Aussagen zu getroffen werden.

	Twitter	**XMPP**	**Amazon SNS**
Pull/Push	Pull und Push	Push	Push
Kommunikations-möglichkeiten und Paradigmen	Pub/Sub und Point-to-Point	(Gruppen-)Chat, Pub/Sub, Video, Nachrichten, Point-to-Point	Pub/Sub
Anwendung-zu-Anwendung-Kommunikation	Ja	Ja	Ja
Nachrichten-formate	JSON, XML, RSS, ATOM, HTTP Get/Post Parameter	XML	Text, JSON, HTTP Post Parameter
Transport-protokolle	HTTP, HTTPS	TCP, HTTP und andere möglich	HTTP, HTTPS, SMS, SMTP
Fat Ping/Light Ping	Light	Fat oder Light	Fat/Light
Nachrichten-begrenzung	140 Zeichen	keine Angaben	8 KB, bei SMS 70-140 Zeichen (je nach Kodierung)
Kommunikations-partner	Dienstnutzer-Dienstgeber	Client-Server, bei Erweiterung auch Peer-to-Peer	Dienstnutzer-Dienstgeber
Kommunikations-richtung	keine Einschränkung	One-Way	keine Einschränkung

Tabelle 2: Einteilung der Technologien nach Eigenschaften der Übertragung (Teil 1)

[35] vgl. [2, S. 497]

	Amazon SQS	Android C2DM	Apple Push Notification
Pull/Push	Pull	Push	Push
Kommunikations-möglichkeiten und Paradigmen	Point-to-Point, Many-to-Many, One-to-Many	Point-to-Point	Point-to-Point
Anwendung-zu-Anwendung-Kommunikation	Ja	Ja	Nein
Nachrichten-formate	XML, HTTP Get/Post Parameter	HTTP Get/Post Parameter, Key-Value Pairs auf der Androidplattform	Bytestrom, Payload als JSON
Transport-protokolle	HTTP, HTTPS	HTTPS	TCP Streaming Sockets
Fat Ping/Light Ping	Fat	Light	Light
Nachrichten-begrenzung	64 KB	1024 Bytes	256 Bytes
Kommunikations-partner	Dienstnutzer-Dienstgeber	Dienstnutzer-Dienstgeber	Dienstnutzer-Dienstgeber
Kommunikations-richtung	keine Einschränkung	One-Way	One-Way

Tabelle 3: Einteilung der Technologien nach Eigenschaften der Übertragung (Teil 2)

4.3 Sicherheit

Hinsichtlich der Kommunikation konnten im Bereich Sicherheit einige Dimensionen ermittelt werden, die in Tabelle 4 dargestellt sind[36]. Dabei wurden eine Reihe von Authentifizierungsverfahren identifiziert, die in Tabelle 4 festgehalten wurden. Ohne hier auf jedes Verfahren tiefer eingehen zu müssen, lässt sich ableiten, dass nur eine Minderheit auf Verfahren mit Benutzernamen und Passwörter setzt, inbesondere wenn bei dem System Anwendungen von Drittanbietern zum Einsatz kommen. Dies sollte bei einem Systementwurf berücksichtig werden, wobei einige Vor- und Nachteile diesbezüglich schon im Abschnitt 3.2.2 erwähnt wurden. Des Weiteren bieten alle Technologien, auch wenn es bei einigen nur optional ist, die Möglichkeit zur Verschlüsselung der Übertragung. Dies ist wichtig um eine

[36] Es konnten auch andere Dimensionen, wie Anfälligkeit für Spam in dezentralisierten Publish-Subscribe Systemen oder Echtheit der Identität identifziert werden, welche allerdings nicht berücktsichtig wurden, da der Fokus auf der Kommunikation liegt.

Reihe von Angriffen vorzubeugen. Allerdings bleibt die Problematik, dass Anwendungen die über zentrale Systeme kommunizieren, jeweils zu dem zentralen Dienst eine veschlüsselte Verbindung haben, aber die Daten innerhalb dieser Verbindung unverschlüsselt sind und somit von dem zentralen System gelesen werden können. So wäre es z.B. möglich, für einen Administrator eines XMPP-Servers die Kommunikation zwischen zwei Eintitäten zu verfolgen [17, S. 194]. Dies kann mittels einer Ende-zu-Ende Verschlüsselung umgangen werden, was allerdings keine der Technologien von Grund auf bietet. Somit sollten bei sensiblen Daten entsprechende Verfahren genutzt werden, die dies bewerkstelligen. Des Weiteren ist zu beachten, wenn diese Dienste Daten zwischenspeichern (wie beispielsweise Amazon SQS), dass diese Daten u.U. unverschlüsselt gespeichert werden. Bei Diensten wie C2DM, wo der eigentliche Nutzdatenteil in der Regel nicht über den Service übertragen wird, hat die Ende-zu-Ende Verschlüsselung möglicherweise eine geringere Relevanz, da sensible Daten über eine eigene Verbindung übertragen werden, die unabhängig von dem Dienst sind.

	Twitter	**XMPP**	**Amazon SNS**	**Amazon SQS**	**Android C2DM**	**Apple Push Notification**
Authentifizierungsverfahren	OAuth (in Ausnahmefällen Basic HTTP)	SASL	Signaturen über HMAC-SHA Hash-Funktion	Signaturen über HMAC-SHA Hash-Funktion	E-Mail-adresse und Token	Zertifikate und Token
Ende-zu-Ende Verschlüsselung	Nein	Nein	Nein	Nein	Nein	Nein
Verschlüsselung der Übertragung	SSL	TLS	SSL	SSL	SSL	TLS oder SSL

Tabelle 4: Dimensionen, die sich zu den Sicherheitsaspekten, der behandelten Technologien ergeben

4.4 Sonstiges

Der Kategorie Sonstiges, welchen in Tabelle 5 dargestellt ist, wurden die Dimensionen Plattformeinschränkungen, Kosten und Vorrausetzungen zugeteilt. Kosten sollten in diesem Fall nicht unbedingt als ein Nachteil gesehen werden, da das komplette Angebot berücksichtig werden muss. Bei einem Angebot im Web, welches zwar Kostenpflichtig ist, aber dies dafür mit besonderen Serviceleistungen ausgleicht, kann dem Kunden weitere Kosten für Personal, Hardware oder Entwicklung sparen. Die Plattformbeschränkung kann eine Einschränkung der Zielgruppe oder der Bedarf weiterer technischer Lösungen bedeuten, welche im Vorfeld abgewogen werden sollte. Die Ausprägungen unter Vorrausetzung können hier eher als Hinweis verstanden werden, welche man bei der Entwicklung beachten sollte.

	Twitter	**XMPP**	**Amazon SNS**	**Amazon SQS**	**Android C2DM**	**Apple Push Notification**
Plattformbeschränkung	keine	keine	keine	keine	Android ab Version 2.3	iOS 3.0 and OS X v7.0
Kosten	keine	keine	Ja	Ja	keine	keine
Vorraussetzung	keine	keine	Creditkarte	Creditkarte	Freischaltung durch Google	SSL Zertifikate

Tabelle 5: Sonstige Dimensionen der behandelten Technologien

4.5 Fazit und Zusammenfassung

In dieser Ausarbeitung wurden grundsätzliche Begriffe der Domäne behandelt und erklärt. Danach wurden asynchrone Echtzeitkommunikationssysteme besprochen. Mit diesem Kenntnisstand wurden dann Dimensionen abgeleitet und Ausprägungen der einzelnen Systeme für die Dimensionen bestimmt. Mit den identifizierten Dimensionen und den gesammelten Ausprägungen ist es nun möglich, weitere Überlegungen zu tätigen und weitere Informationen zu sammeln, um Archtitekturabwägungen vorzunehmen oder in Ansätzen eigene Architekturkonzepte zu entwerfen.

4.6 Ausblick

Die Technologien pubsubhubbub[37] und Java Messaging Service (JMS) erscheinen für die Thematik dieser Ausarbeitung als relevant, konnten aber aus zeitlichen Gründen nicht mehr betrachtet werden. Des Weiteren konnten auch die verschieden Authentifizierungsverfahren nur angeführt und nicht detailliert beschrieben werden. Eine nähere Betrachtung dieser Technologien, sowie ein Vergleich der verschiedenen Authentifizierungsverfahren, können den hier gemachten Überblick weiter ausbauen. Ein weiteres interessantes Feld ist Betrachtung der verschieden Verbindungsarten zur Umsetzung des Push-Verfahrens, wie TCP oder langlebige HTTP-Verbindungen. Die Unterschiede dieser Verfahren gilt es herauszuarbeiten, um weitere Dimensionen zu entdecken und Vor- wie Nachteile benennen zu können. Des Weiteren fehlt noch eine detailliertere Auseinandersetzung mit den möglichen Vor- und Nachteilen der einzelnen Ausprägung. Damit wurde im letzten Kapitel dieser Ausarbeitung zwar schon begonnen, allerdings wurde dies noch nicht erschöpfend behandelt. Zuletzt sei angemerkt, dass auch andere Perspektiven eingenommen werden sollten, die weniger kommunikationsorientiert sind, um weitere Dimensionen zu identifizieren.

[37] siehe http://code.google.com/p/pubsubhubbub/

Literatur

[1] Martin Böhringer and Peter Gluchowski. Microblogging. *Informatik-Spektrum*, 32:505–510, 2009. http://dx.doi.org/10.1007/s00287-009-0383-0.

[2] C.D. E and D.E. Comer. *TCP/IP- Studienausgabe: Konzepte, Protokolle, Architekturen*. mitp Professional. mitp/bhv, 2011.

[3] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35:114–131, June 2003.

[4] Aleksandar Gargenta. Mastering c2dm the android cloud to device messaging framework. https://docs.google.com/present/edit?id=0AeQcLGA4QADQZHZwNjVkaF8zMmNncWt6Y2Rt&hl=en, 2011, Letzte Sichtung am 13.1.2012.

[5] Aleksandar Gargenta. Screencast: Mastering c2dm the android cloud to device messaging framework. http://www.youtube.com/watch?v=dWwrNTG7dgc, 2011, Letzte Sichtung am 30.1.2012.

[6] A. Gulbrandsen. Rfc2782 - a dns rr for specifying the location of services (dns srv). http://tools.ietf.org/html/rfc2782l, 2000, Letzte Sichtung am 23.02.2012.

[7] Eren Hammer. The oauth 1.0 guide - introduction. http://hueniverse.com/oauth/guide/intro/, Mai 2010, Letzte Sichtung am 14.03.2012.

[8] Eren Hammer. Oauth 1.0. http://hueniverse.com/oauth/, 2010/11, Letzte Sichtung am 14.03.2012.

[9] Apple Inc. Local and push notification programming guide. http://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/RemoteNotificationsPG.pdf, August 2011, Letzte Sichtung am 30.12.2011.

[10] Google Inc. Comparison of pubsubhubbub to light-pinging protocols. http://code.google.com/p/pubsubhubbub/wiki/ComparingProtocols, Februar 2010, Letzte Sichtung am 25.02.2012.

[11] Google Inc. Android cloud to device messaging framework. http://code.google.com/intl/de-DE/android/c2dm/, 2011, Letzte Sichtung am 12.1.2012.

[12] Google Inc. Authentication and authorization for google apis - clientlogin for installed applications. http://code.google.com/intl/de-DE/apis/accounts/docs/AuthForInstalledApps.html, 2011, Letzte Sichtung am 15.1.2012.

[13] Mark Linderman, Norman Ahmed, James Metzler, and Jason Bryant. A hybrid publish subscribe protocol. In *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, Companion '08, pages 24–29, New York, NY, USA, 2008. ACM. http://doi.acm.org/10.1145/1462735.1462742.

[14] Ralph Meijer Peter Millard, Peter Saint-Andre. Xep-0060 - publish-subscribe. http://xmpp.org/extensions/xep-0060.html, Juli 2010, Letzte Sichtung am 28.02.2012.

[15] Peter Saint-Andre. Xep-0174 - serverless messaging. http://xmpp.org/extensions/xep-0174.html, 2008, Letzte Sichtung am 22.02.2012.

[16] Peter Saint-Andre. Rfc6120 - extensible messaging and presence protocol (xmpp) - core. http://xmpp.org/rfcs/rfc6120.html, 2011, Letzte Sichtung am 19.02.2012.

[17] Peter Saint-Andre, Kevin Smith, and Remko Tronçon. *XMPP: The Definitive Guide: Building Real-Time Applications with Jabber Technologies*. O'Reilly Media, Inc., 1 edition, April 2009.

[18] Amazon Web Service. Amazon simple queue service: Developer guide. http://awsdocs.s3.amazonaws.com/SQS/latest/sqs-dg.pdf, Oktober 2011, Letzte Sichtung am 17.07.2012.

[19] Amazon Web Service. Amazon simple queue service: Api reference. http://awsdocs.s3.amazonaws.com/SQS/latest/sqs-api.pdf, Oktober 2011, Letzte Sichtung am 21.03.2012.

[20] Amazon Web Service. Amazon simplenotification service: Api reference. http://awsdocs.s3.amazonaws.com/SNS/latest/sns-api.pdf, März 2012, Letzte Sichtung am 22.03.2012.

[21] A.S. Tanenbaum and M. van Steen. *Distributed systems: principles and paradigms*. Pearson Prentice Hall, Upper Saddle River, 2002.

[22] Twitter. Getting started. https://dev.twitter.com/start, 2011, Letzte Sichtung am 12.03.2012.

[23] Twitter. Things every developer should know. https://dev.twitter.com/docs/things-every-developer-should-know, Juli 2011, Letzte Sichtung am 15.03.2012.

[24] Twitter. Streaming api. https://dev.twitter.com/docs/streaming-api, Januar 2011, Letzte Sichtung am 17.03.2012.

[25] Twitter. Streaming api - concepts. https://dev.twitter.com/docs/streaming-api/concepts, Februar 2012, Letzte Sichtung am 15.03.2012.

A Anhang

A.1 DNS Service Lookup und Auswahl eines XMPP-Dienstes

DNS bietet den Vorteil, dass ein Mensch sich keine IP-Adressen merken muss. Die Domains müssen allerdings entsprechend aufgelöst werden, damit man die IP-Adresse hinter einer Domain erhält und eine TCP-Verbindung aufbaut werden kann[38]. Des Weiteren bietet DNS den Vorteil, dass mehrere Server für eine Domain eingesetzt werden können, somit ist es möglich Dienste unter einer Domain von Host zu Host zu verschieben [6]. Da sich also mehrere Server oder sogar mehrere Dienste unter einer Domain befinden können, muss eine XMPP-Entität zuerst den Fully Qualified Domain Name des Dienstes ermitteln, um die IP-Adresse aus dem FQDN auflösen zu können. Bei XMPP wird für das Ermitteln des FQDN in der Regel erst ein DNS Service Lookup[39] durchgeführt. Dieser besitzt das Format "servicename.protokoll.domain". Der Servicename wird je nach Verbindung, sprich Aufbau einer Client-zu-Server oder Server-zu-Server Verbindung als "xmpp-client" oder "xmpp-server" gewählt. Das Protokoll erhält für diesen Fall den Wert "tcp" und die die Domain den Namen der Domain. Als Beispiel ist ein nslookup-Aufruf[40] mit dem Abfragestring "_xmpp-client._tcp.google.com" in Abbildung 12 dargestellt. Die Domain google.com beinhaltet hierbei mehrere XMPP-Services auf dem gleichen Port 5222 mit dem Übertragungsprotokoll TCP. Eine XMPP-Entität kann nun einen dieser Services über die Priorität (Priority) bzw. der Gewichtung (Weight) auswählen, wobei entweder der Service mit der niedrigsten Priorität oder bei mehreren Services mit der gleichen Priorität, der Service mit der höchsten Gewichtung, die größte Wahrscheinlichkeit erhält, gewählt zu werden. Die IP-Adresse wird dann über einen "A" oder "AAAA" lookup[41] des "srv hostname" (oder auch FQDN) ermittelt und zusammen mit dem Port zum Aufbau der TCP-Verbindung verwendet (siehe Abbildung 13).

Sollten mehrere IP-Adressen für den "A" bzw. "AAAA" lookup zurück geben werden und der Verbindungsversuch mit einer der Adresse schlägt fehl, so wird die nächste IP-Adresse gewählt. Schlagen alle IP-Adressen für diese FQDN fehl, so wird der nächste FQDN nach den oben beschrieben Kriterien verwendet.

Für den Fall, dass schon der erste DNS Service Lookup keine Ergebnisse liefert, wird ein "A" oder "AAAA" lookup auf die ursprüngliche Domain durchgeführt (in diesem Beispiel google.com) und mittels der resultierenden IP-Adresse versucht, über den Standardport 5222 für Clients oder 5269 für Server, eine Verbindung aufzubauen [16, Abschnitt 3.2.1-3.2.2].

[38] Für diese Seminararbeit wird nur TCP als Übertragungsprokoll betrachtet.
[39] Dies bedeutet, dass der Service Resource Record (SRV record) wird durchsucht
[40] Alternativ kann auch der dig-Befehl (Domain Information Groper) verwendet werden.
[41] Zum Aufösen einer IPv4- oder IPv6-Adresse

```
C:\>nslookup -type=SRV _xmpp-client._tcp.google.com
Server:  netconnect.box
Address:  192.168.0.1

Nicht autorisierende Antwort:
_xmpp-client._tcp.google.com    SRV service location:
          priority       = 20
          weight         = 0
          port           = 5222
          svr hostname   = alt4.xmpp.l.google.com
_xmpp-client._tcp.google.com    SRV service location:
          priority       = 5
          weight         = 0
          port           = 5222
          svr hostname   = xmpp.l.google.com
_xmpp-client._tcp.google.com    SRV service location:
          priority       = 20
          weight         = 0
          port           = 5222
          svr hostname   = alt1.xmpp.l.google.com
_xmpp-client._tcp.google.com    SRV service location:
          priority       = 20
          weight         = 0
          port           = 5222
          svr hostname   = alt2.xmpp.l.google.com
_xmpp-client._tcp.google.com    SRV service location:
          priority       = 20
          weight         = 0
          port           = 5222
          svr hostname   = alt3.xmpp.l.google.com
```

Abbildung 12: Ausschnitt einer Ausgabe eines nslookup Befehls

```
C:\>nslookup xmpp.l.google.com
Server:  netconnect.box
Address:  192.168.0.1

Nicht autorisierende Antwort:
Name:    xmpp.l.google.com
Addresses:  2a00:1450:400c:c01::7d
          173.194.70.125
```

Abbildung 13: Auflösung des FQDN zu einer IP-Adresse mittels nslookup

A.2 Veröffentlichen einer Nachricht mit Payload bei XMPP

Algorithmus 8 Veröffentlichen einer Nachricht mit Payload modifiziert aus [14, Abschnitt 7.1.1]

```
<iq type='set'
    from='publisher@example.org/home-pc'
    to='pubsub.example.org'
    id='publish1'>
  <pubsub xmlns='http://jabber.org/protocol/pubsub'>
   <publish node='TestNode'>
     <item id='fwelp43gsdg4'>
       <entry xmlns='http://www.w3.org/2005/Atom'>
         <title>Example Text</title>
         <summary>Some Text</summary>
          <link rel='alternate' type='text/html'
                href='http://example.org/2012/03/25/atom01'/>
                  <id>tag:example.org,2012:entry-11</id>
          <published>2012-03-25T18:30:02Z</published>
          <updated>2012-03-25T18:30:02Z</updated>
                </entry>
      </item>
    </publish>
  </pubsub>
</iq>
```